

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования

«Национальный исследовательский университет ИТМО»

Факультет Программной Инженерии и Компьютерной Техники

Дисциплина: Тестирование программного обеспечения

Лабораторная работа 1

Вариант 22053

Выполнил:

Гурьянов Кирилл Алексеевич

Группа: Р33302

Преподаватель:

Гаврилов Антон Валерьевич

Санкт-Петербург

2024

Задание

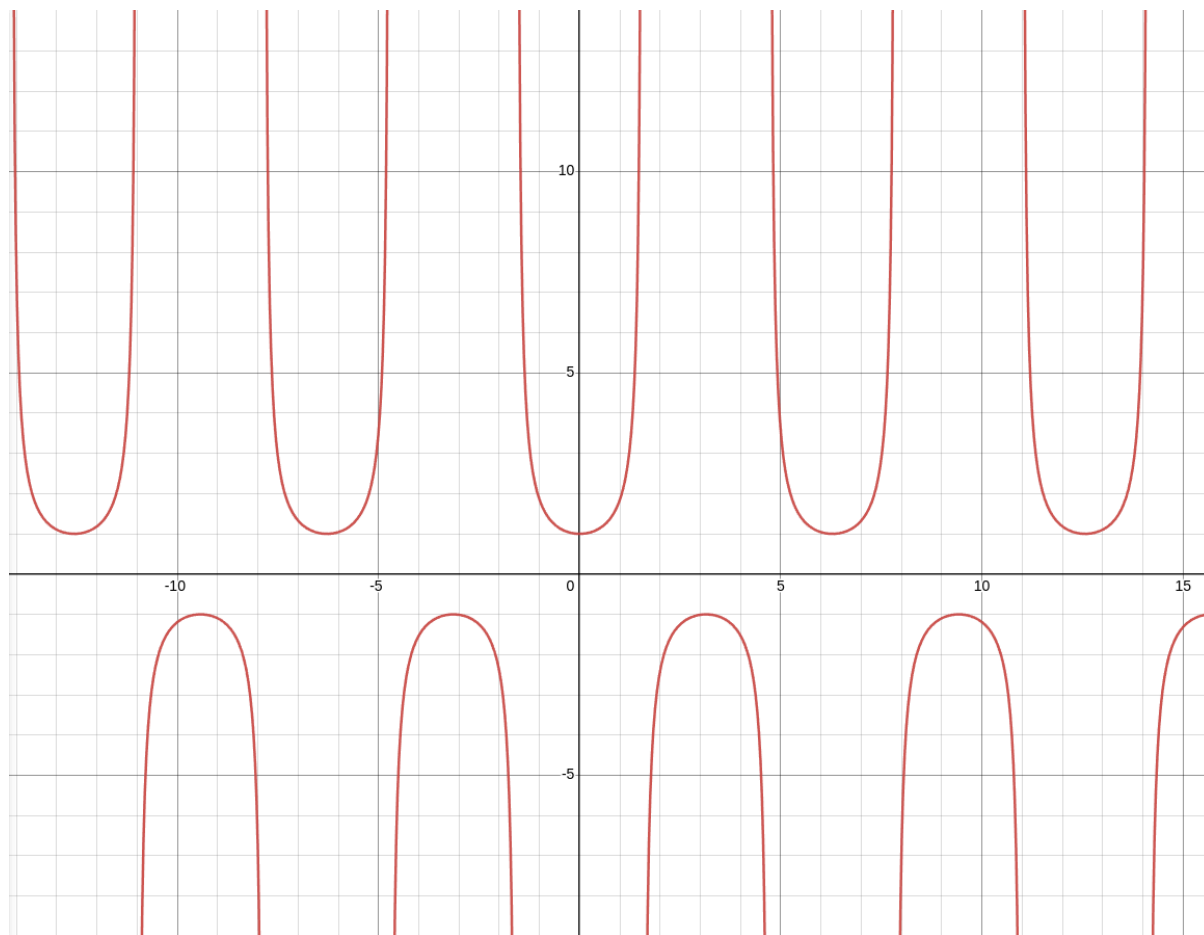
1. Для указанной функции провести модульное тестирование разложения функции в степенной ряд. Выбрать достаточное тестовое покрытие.
2. Провести модульное тестирование указанного алгоритма. Для этого выбрать характерные точки внутри алгоритма, и для предложенных самостоятельно наборов исходных данных записать последовательность попадания в характерные точки. Сравнить последовательность попадания с эталонной.
3. Сформировать доменную модель для заданного текста. Разработать тестовое покрытие для данной доменной модели

Вариант 22053

1. Функция $\sec(x)$
2. Программный модуль для работы с хеш-таблицей с разрешением коллизий методом цепочек (Hash Integer, <http://www.cs.usfca.edu/~galles/visualization/BucketSort.html>)
3. Описание предметной области:
После того, как толпа вновь разразилась ликующими криками, Артур обнаружил, что он скользит по воздуху к одному из величественных окон во втором этаже здания, перед которым стоял помост, с которого оратор обращался к народу.

1 задание

График



Разложение функции в степенной ряд

$$\sec(x) = 1 + \frac{1}{2}x^2 + \frac{5}{24}x^4 + \frac{61}{720}x^6 + \dots = \sum_{n=0}^{\infty} \frac{(-1)^n E_{2n}}{(2n)!} x^{2n}, \text{ для всех } |x| < \frac{\pi}{2},$$

где E_{2n} — числа Эйлера

Числа Эйлера

Эйлеровы числа — целые числа E_0, E_1, E_2, \dots , использующиеся при разложении гиперболического секанса в степенной ряд.

$$\frac{1}{\operatorname{ch}(t)} = \sum_{n=0}^{\infty} \frac{E_n t^n}{n!}$$

Так как функция $\operatorname{ch}(t)$ четная, то $E_1 = E_3 = E_5 = \dots = E_{2n+1} = \dots = 0$

Четные числа Эйлера могут быть получены по формуле:

$$E_{2n} = - \sum_{k=0}^{n-1} \frac{(2n)!}{(2k)!(2n-2k)!} E_{2k}$$

При этом $E_0 = 1$, таким образом первые числа Эйлера имеют вид:

$$E_2 = -1, E_4 = 5, E_6 = -61, E_8 = 1385, E_{10} = -50521$$

Реализация подсчета чисел Эйлера

```
public double calcEulerNumber(int m) {
    if (m < 0) throw new IllegalArgumentException("n must be non-negative");
    if (m > 60) throw new IllegalArgumentException("n is too large");
    if (m == 0) return 1;
    if (m % 2 != 0) return 0;
    int n = m / 2;
    double E = 0;
    for (int k = 0; k <= n - 1; k++) {
        E += CombinatoricsUtils.binomialCoefficient(2 * n, 2 * k) *
calcEulerNumber(2 * k);
    }
    E *= -1;
    return E;
}

package functions;

import util.MathUtils;

public class Sec {
    MathUtils mathUtils = new MathUtils();

    public double sec(double x, int n) throws IllegalArgumentException {
        if (n > 30) throw new IllegalArgumentException("The decomposition order
is too large");
        double sec = 0;
        int sign = 1;
        x = bringXtoFirstTurnOfCircle(x);
        if (Math.abs(x) > Math.PI / 2) {
            if (x > 0) x = -Math.PI + x;
            else x = Math.PI + x;
            sign = -1;
        }
        double eps = 0.0999;
        if (Math.abs(x - Math.PI / 2) < eps || Math.abs(x + Math.PI / 2) < eps)
            throw new IllegalArgumentException("The function is not defined for
this value");
        double factorial = 1;
        int t = 1;
```

```

    for (int k = 0; k < n; k++) {
        if (k != 0) {
            factorial /= t;
            factorial /= t + 1;
            t += 2;
        }
        double E = mathUtils.calcEulerNumber(2 * k);
        sec += Math.pow(-1, k) * E * Math.pow(x, 2 * k) * factorial;
    }
    return sec * sign;
}

private double bringXtoFirstTurnOfCircle(double x) {
    double doublePI = 2 * Math.PI;
    while (x > Math.PI)
        x = x - doublePI;
    while (x < -Math.PI)
        x = x + doublePI;
    return x;
}
}

```

Функция $y = \sec(x)$ не определена в точках $x = \frac{\pi}{2} + \pi k, k \in \mathbb{Z}$

Однако у разложения функции $\sec(x)$ в ряд степенной ряд ОДЗ: $|x| < \frac{\pi}{2}$

Таким образом, сначала точка приводится к первому обороту круга, т.е. к отрезку $[-2\pi; 2\pi]$, а после этого по формулам приведения уже к интервалу $(-\frac{\pi}{2}; \frac{\pi}{2})$

В ходе тестирования было выявлено, что вычисление значения функции в степенной ряд имеет существенную погрешность при x близких к $x = \frac{\pi}{2} + \pi k, k \in \mathbb{Z}$, причем чем ближе x к точки неопределенности, тем большую погрешность дает разложение. В связи с чем было решено ограничить область допустимых значений нашей функции множеством:

$$x \in \left[-\frac{\pi}{2} + 0.0999 + 2\pi k; \frac{\pi}{2} - 0.0999 + 2\pi k\right] \cup \left[\frac{\pi}{2} + 0.0999 + 2\pi k; \frac{3\pi}{2} - 0.0999 + 2\pi k\right], k \in \mathbb{Z}$$

Модульное тестирование разложения функции в степенной ряд

Для начала было создано тестовое покрытие для функции вычисления чисел Эйлера, которое включало в себя:

- Тест на четные числа Эйлера
- Тест на нечетные числа Эйлера
- Тест на слишком большие числа Эйлера
- Тест на отрицательные числа Эйлера

Тестовое покрытие для функции секанса включил в себя тесты:

- Тест на точки внутри промежутка $[0; 2\pi]$ входящие в ОДЗ
- Тест на точки внутри промежутка $[-2\pi; 0]$ входящие в ОДЗ
- Тест на точки неопределенности $x = \frac{\pi}{2} + 2\pi k, k \in \mathbb{Z}$
- Тест на точки, близкие к точкам неопределенности и не отходящие от точек $x = \frac{\pi}{2} + 2\pi k, k \in \mathbb{Z}$ более чем на $\epsilon < 0.0999$
- Тест на точки, близкие к точкам неопределенности и отдаленные от них на $\epsilon \geq 0.0999$
- Тест на точки, чем модуль больше 2π

Код тестов

<https://github.com/Korako2/TPO-LAB1/blob/master/src/test/java/functions/SecTest.java>

<https://github.com/Korako2/TPO-LAB1/blob/master/src/test/java/util/MathUtilsTest.java>

2 задание

Реализация HashTable

```
package algorithm;

import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;

public class HashTable<V> {
    private class Entry {
        Integer key;
        V value;
    }
}
```

```

    Entry(Integer key, V value) {
        this.key = key;
        this.value = value;
    }
}

private final List<List<Entry>> buckets;

public HashTable(int nBuckets) {
    if (nBuckets <= 0)
        throw new IllegalArgumentException("Number of buckets must be
greather than 0");

    this.buckets = new ArrayList<>(nBuckets);

    for (int i = 0; i < nBuckets; i++) {
        this.buckets.add(new LinkedList<>());
    }
}

public V put(Integer key, V value) {
    int hashKey = hashKey(key);
    List<Entry> bucket = buckets.get(hashKey);

    for (Entry entry : bucket) {
        if (entry.key.equals(key)) {
            V prev = entry.value;
            entry.value = value;
            return prev;
        }
    }

    Entry newEntry = new Entry(key, value);
    bucket.add(newEntry);

    return null;
}

public V get(Integer key) {
    int hashKey = hashKey(key);
    List<Entry> bucket = buckets.get(hashKey);
    return bucket.stream().filter(e -> e.key.equals(key)).map(e ->
e.value).findFirst().orElse(null);
}

public V delete(Integer key) {
    int hashKey = hashKey(key);
    List<Entry> bucket = buckets.get(hashKey);
    for (int i = 0; i < bucket.size(); i++) {
        if (bucket.get(i).key.equals(key)) {
            V prev = bucket.get(i).value;

```

```

        bucket.remove(i);
        return prev;
    }
}

return null;
}

private int hashCode(Integer key) {
    return Math.abs(key) % buckets.size();
}
}

```

Модульное тестирование указанного алгоритма

Тестовое покрытие для алгоритма включает в себя:

- Запрещенное количество блоков
- Добавление единичного элемента
- Добавление нескольких элементов по одному ключу
- Добавление нескольких элементов с коллизиями
- Попытка достать элемент по несуществующему ключу
- Проверка удаления элемента по ключу
- Попытка удаления элемента по несуществующему ключу
- // Повторная попытка удаления

Реализация тестового покрытия

```

package algorithm;

import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.ValueSource;

import static org.junit.jupiter.api.Assertions.*;

class HashTableTest {
    @ParameterizedTest
    @DisplayName("Illegal bucket number")
    @ValueSource(ints = {0, -1})
    public void illegalBucketNumber(int n) {
        assertThrows(IllegalArgumentException.class, () -> new
        HashTable<String>(n));
    }

    @Test

```



```

@Test
@DisplayName("Insertion of a single element")
public void insertion() {
    HashTable<String> table = new HashTable<>(5);
    assertNull(table.put(0, "Ivan Varyukhin"));
    assertEquals("Ivan Varyukhin", table.get(0));
}

@Test
@DisplayName("Repeated insertion of the same key")
public void repeatedInsertion() {
    HashTable<String> table = new HashTable<>(5);
    assertNull(table.put(0, "Ivan Varyukhin"));
    assertEquals("Ivan Varyukhin", table.get(0));
    assertEquals("Ivan Varyukhin", table.put(0, "Arkhip Ryabokon"));
    assertEquals("Arkhip Ryabokon", table.get(0));
}

@Test
@DisplayName("Insertion with a collision")
public void collisionsInsertion() {
    String[] names = new String[]{"Ivan Varyukhin", "Arkhip Ryabokon",
    "Chernova Elizabeth", "Kanukova Eva", "Pavel Solovyov", "Malika Agadilova"};
    // Number of buckets = 5, inserting 6 elements will guarantee a collision
    HashTable<String> table = new HashTable<>(5);
    for (int i = 0; i < 6; i++) {
        assertNull(table.put(i, names[i]));
        for (int j = 0; j <= i; j++) {
            assertEquals(names[j], table.get(j));
        }
    }
}

@Test
@DisplayName("Get the value from a non-existent key")
public void getNonExistentKey() {
    HashTable<String> table = new HashTable<>(5);
    assertNull(table.get(52));
}

@Test
@DisplayName("Delete a key")
public void deleteKey() {
    HashTable<String> table = new HashTable<>(5);
    assertNull(table.put(0, "Ivan Varyukhin"));
    assertEquals("Ivan Varyukhin", table.get(0));
    assertEquals("Ivan Varyukhin", table.delete(0));
    assertNull(table.get(0));
}

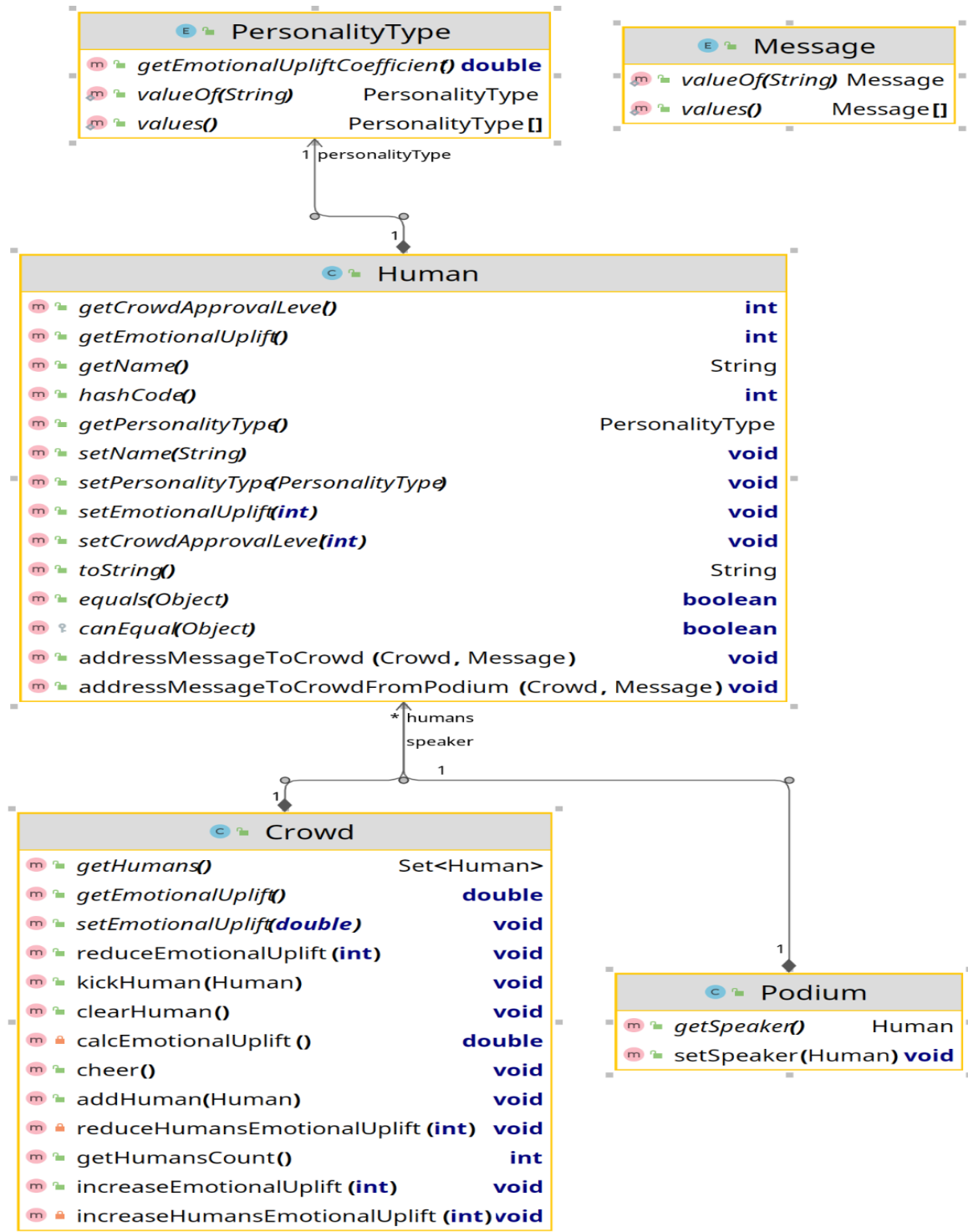
@Test
@DisplayName("Delete a non-existent key")

```

```
public void deleteNonExistentKey() {  
    HashTable<String> table = new HashTable<>(5);  
    assertNull(table.delete(0));  
}  
}
```

3 задание

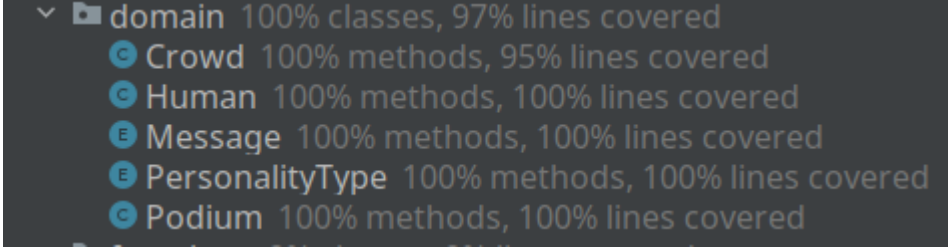
Разработанная доменная модель



Реализация тестового покрытия

<https://github.com/Korako2/TPO-LAB1/blob/master/src/test/java/domain/SceneTest.java>

Процент покрытия тестами



A screenshot of a code coverage report for the 'domain' package. The report shows the following coverage statistics:

Package/Class	Methods Covered	Lines Covered
domain	100%	97%
Crowd	100%	95%
Human	100%	100%
Message	100%	100%
PersonalityType	100%	100%
Podium	100%	100%

Вывод

В ходе выполнения лабораторной работы мною были изучены основы модульного тестирования, проведено тестирование тригонометрической функции, собственной реализации хэш-таблицы, а также доменной модели по данному тексту.